Dynamics of Agile Software Development

Kim E. van Oorschot
Eindhoven University of Technology
P.O. Box 513, Pav. M0.06, 5600 MB
Eindhoven, The Netherlands
T +31 40 247 4435
F +31 40 246 8054
k.e.v.oorschot@tue.nl

Kishore Sengupta
kishore.sengupta@insead.edu
INSEAD, Fontainebleau, France

Luk N. van Wassenhove
luk.van-wassenhove@insead.edu
INSEAD, Fontainebleau, France

Abstract

Software projects have traditionally been problematic in terms of quality, cost and time. Researchers and practitioners have focused on agile software development as an alternative to overcome these problems. Agile methods employ iterative development cycles (typically 20 working-days), interspersed by user feedback. The key to agile projects is the sense of urgency created by the need to deliver at regular intervals. This paper examines this construct, i.e., schedule pressure. We investigate the relationship between the level of agility (length of the iterative cycle) and project outcomes. We argue that project outcomes may suffer either from a team being too inactive, e.g., in sequential or low levels of agility, or from a team being over-active over too long, a situation likely to occur in high levels of agility. We hypothesize that moderate levels of agility are likely to result in the best project outcomes. We test our hypothesis through simulation, and find a U-shaped pattern: performance is better when iteration lengths are 50 working-days, as opposed to 20 working-day cycles widely used in practice. Our analysis provides both theoretical insights into the dynamics of agile software development and practical suggestions for managing these projects.

Key words: schedule pressure, agile software development, iterations, system dynamics

1 Introduction

Software is a core component in a wide range of products ranging from mobile phones to automobiles, health care and financial services. Therefore, the effectiveness with which software development projects are managed influences the quality of products and services offered by organizations. However, the reality of software projects is that they frequently run late, over budget and suffer from problems in quality and functionality (Austin 2001; Lindstrom and Jeffries 2004, Molokken-Østvold and Jørgensen 2005; Meso and Jain 2006; Cao and Ramesh 2008).

Agile or iterative software development methods are increasingly being adopted as an alternative to traditional methods. Agile methods are: "…lightweight processes that employ short iterative cycles, actively involve users to establish, prioritize, and verify requirements, and rely on a team's tacit knowledge as opposed to documentation. A truly agile method must be *iterative* (take several cycles to complete), *incremental* (not deliver the entire product at once), self-organizing (teams determine the best way to handle work), and emergent (processes, principles, and work structures are recognized during the project rather than predetermined)" (Boehm and Turner 2005, p. 32). Typically, an agile project is divided into several parts or iterations. In each iterative cycle – typically one month (20 working days) - a part of the complete product is finished and shown to the customer. Based on the customer's feedback, the team and the customer decide on the deliverables to develop in the next iterative cycle (Schatz and Abdelshafi 2005; Larman 2006).

These iterative cycles create deadlines that are never distant in time. Deadlines force progress, and thus teams always work with a sense of urgency, rather than being comparatively inactive in the beginning and then experiencing massive schedule pressure toward the end. Customers provide frequent feedback, and thus little effort is wasted on implementing obsolete or incorrect specifications. Evidence on the effectiveness of agile

methods – while nascent and mostly anecdotal – suggests that they lead to: improved accuracy of estimates and lower effort overruns (Molokken-Østvold and Jørgensen 2005), reduced time to market (Schatz and Abdelshafi 2005), and fewer defects (Lindstrom and Jeffries 2004; Schatz and Abdelshafi 2005).

This paper focuses on a fundamental aspect of agile methods: that of ensuring progress by creating a sense of urgency throughout the project. The underlying mechanism is schedule pressure (Austin 2001), which is a function of the team's perception of the time required to finish the tasks at hand versus the actual time remaining. Specifically, we examine the relationship between schedule pressure and the *level of agility* as characterized by the length of the iteration, and its impact on project outcomes. We also investigate how this relationship is affected when customers change requirements between iterations.

We hypothesize that at one end of the continuum, i.e., sequential projects, the eventual peak of schedule pressure is so high as to result in sharp increases in errors generated, with not enough time to detect and fix them. The result is not only poor quality, but also extra effort in fixing the errors that *are* detected, as well as time overruns. At the other end of the continuum, i.e., high levels of agility, frequent due dates at the end of each cycle create high peaks of schedule pressure, with an outcome of poor quality. Thus, it is possible to push a team over the edge by planning so many intermediate due dates that there is little no time to recover from the pressure of the previous iteration. That is, agility brings benefits to a project only when the peaks and troughs of schedule pressure are moderate. The hypothesis is tested through a simulation using a widely validated systems dynamics model.

As hypothesized, we find a U-shaped relationship between schedule pressure, project outcomes (in terms of errors, time, costs), and the level of agility, i.e., iteration length. Our results show that the best iteration lengths are closer to 50 working days (2.5 – 3 calendar months) rather than the 20 working day monthly cycle used in the field. Thus, contrary to

widespread practice, there is no "one size fits all" length of the iterative cycle in agile development.

Beyond the prescriptions and conventions of monthly cycles, there is little understanding on how the dynamics of agility affects the behavior of project teams and outcomes. This constitutes a significant gap in our knowledge of agile development, and is of relevance both conceptually and for practitioners. Our research addresses this gap by suggesting a way to think about iteration lengths in terms of schedule pressure, and thus project outcomes.

The paper is structured as follows. We start with a description of the model structure in Section 2, followed by a description of the research method. In Section 4 the results of our simulations are discussed. The paper ends with a discussion of the implications of our study.

2 Literature and Model Structure

2.1 Schedule Pressure

The core concept of our model is *schedule pressure*. Schedule pressure is a widely documented phenomenon in software and product development projects (Perlow 1999; Schatz and Abdelshafi 2005). The project team feels schedule pressure when the time available is inadequate, i.e., the ratio of the number of days required to the number of days available exceeds 1 (Abdel-Hamid and Madnick 1991; Austin 2001).

There are two main reasons for schedule pressure in software projects. Software development is not deterministic. It is extremely difficult to estimate up-front the number of tasks and effort need to complete the tasks (Brooks 1979; Boehm 1981; Abdel-Hamid and Madnick 1991). Moreover, estimates are typically asymmetrical: project size and effort are usually *under*-estimated. This occurs because managers focus on the highly visible mainline components, thereby underestimating or even completely missing the less visible components (e.g. help message processing, or error processing). Moreover, new tasks are discovered

during execution (Abdel-Hamid and Madnick 1991; Eisenhardt and Tabrizi 1995; Dawson and Dawson 1998; McDermott 1999; Tatikonda and Rosenthal 2000; Van Oorschot et al. 2005; Iansiti and McCormack 1997). The additional tasks create higher workload and thus increased schedule pressure unless team size and/or due date are extended.

Second, even when extra time is available as a contingency for "unforeseen" events, team members often use it by being less productive in the beginning, often for gold-plating, e.g., refining features beyond requirements (Goldratt 1997; Gevers et al. 2001; Latham and Seijts 1999; Buehler et al. 1994; Seers and Woodruff 1997; Karau and Kelly 1992; Lyneis and Ford 2007). This loss of productivity in the beginning impedes progress and eventually increases schedule pressure.

2.2 Software development

Figure 1 shows a system dynamics model of software development. Here, software development is conceptualized as transforming a stock of tasks that need to be developed to a stock of developed tasks. A task is defined as a set of deliverables, such as lines of code or source instructions. Initially, most of tasks that need to be developed are known (initial level of *Tasks to be Developed*, Figure 1). The stock *Tasks Developed* is empty, since nothing has been developed as yet.

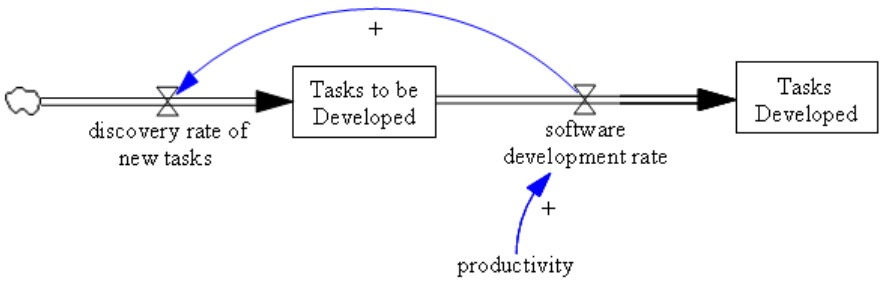

Figure 1: Basic stocks and flows of software development

Once completed, a task flows from *Tasks to be Developed* to *Tasks Developed*. The *software development rate* is the speed at which tasks flow to *Tasks Developed*. This rate is

determined by the team's productivity, i.e., the number of tasks developed in a normal working day. The higher the productivity, the more tasks flow to *Tasks Developed*, and the sooner the project is finished.

The team's workload increases with discovery of new tasks while executing known tasks (Figure 1, *software development rate* → *discovery rate of new tasks*), i.e., the level of *Tasks to be Developed* increases. The project is complete when there are no tasks left in *Tasks to be Developed*, and all tasks are in *Tasks Developed.*

2.3 Consequences of schedule pressure on software development

Higher levels of agility mean more iterations, and thus more reviews and due dates. Now consider how the length of the iteration affects schedule pressure (Figure 2). Sequential development has only one final due date. The team works toward this due date. Schedule pressure exceeds 1 when the team realizes it is running out of time. To compensate, the team starts working overtime. This continues until all tasks are developed or until schedule pressure drops below 1.By working overtime, the team increases its productivity. In agile development, on the other hand, there are *multiple* pressure peaks toward the end of the iterations rather than one large peak toward the end of the project. Thus, agility influences the distribution of schedule pressure and overtime during a project's life cycle. To understand how the number of pressure peaks influences project outcomes, we need to consider the three main effects of schedule pressure described in extant research: overtime and learning, exhaustion and turnover, and errors and rework.

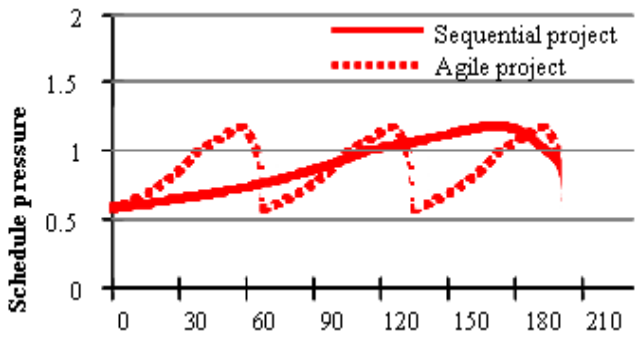

Figure 2: Schedule pressure in sequential and agile projects

2.3.1 Overtime and learning

A team increases its rate of development through overwork (Boehm 1981; Abdel-Hamid and Madnick 1991; Repenning and Sterman 2002; Perlow 1999). When no new tasks are discovered, overtime has a balancing effect on schedule pressure, because a higher development rate decreases the number of tasks remaining (the overwork loop in Figure 3).

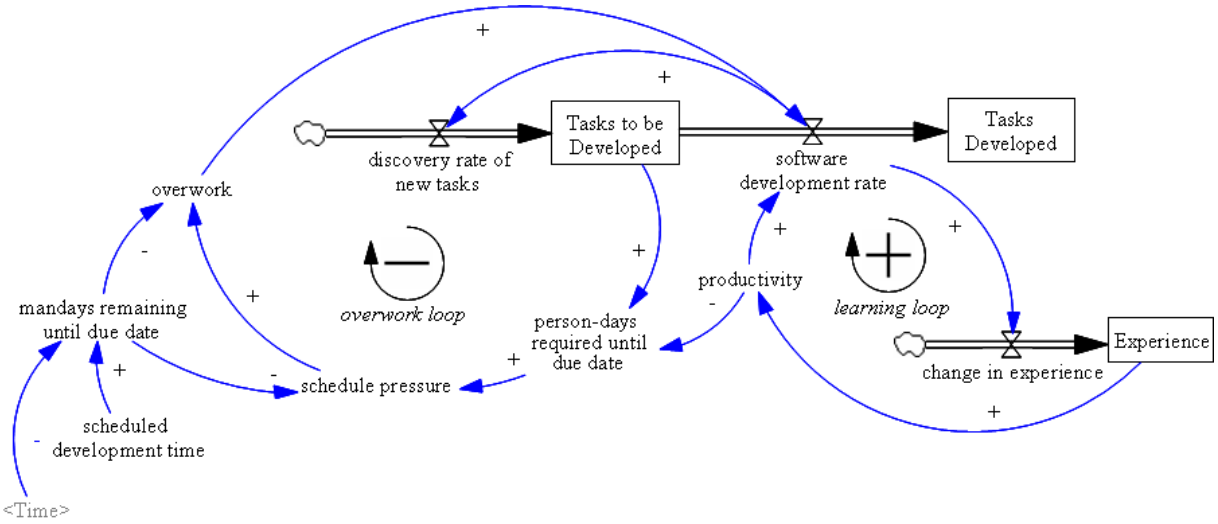

Figure 3: Positive effects of overwork

The level of experience determines team productivity via the learning curve (Sterman 2000; Boh et al. 2007; Ford and Sterman 1998; Kessler and Bierly 2002; Wiersma 2007). At the start of a project, the team is inexperienced and at the bottom of the curve. Experience grows each day the team works on the project. Thus, experience (and therefore productivity) is higher at the end of an iteration than at the beginning. Overwork increases the output of the team, and therefore the team is able to gain more experience. By getting additional experience through more hours spent, the team learns faster (the learning loop in Figure 3). Thus, overwork can increase *software development rate* in the short-term by increasing the number of hours worked per day and in the longer-term by improving the team's experience and productivity.

Due dates at the end of each iteration (or the end of the project in sequential development) *disrupt* continuity in the experience formation cycle. *Before the review*, the

team has to stop working on development tasks in order to prepare. The normal pace of work is also disrupted *during the review*. *After the review*, it takes time for the team to close their mental accounts and decompress before getting ready for the next iteration, i.e., open the next mental account (Thaler 1999).

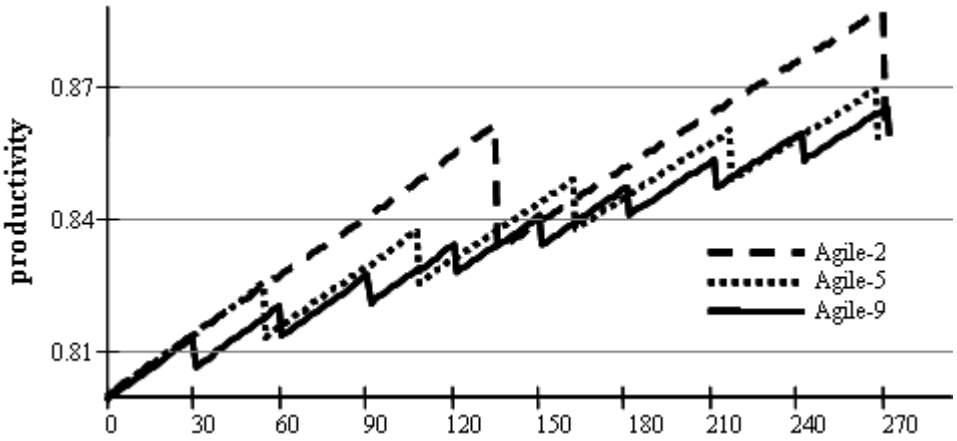

Figure 4: Due dates cause disruption in learning and productivity in agile projects

These disruptions affect the team's productivity (Seshadri and Shapira 2001). Thus, team productivity at the end of iteration i is higher than at the start of iteration $i + 1$. However, because the drop in productivity is less than the productivity gained (via an increase of experience) during the previous iteration, average productivity increases during the project. This behavior is shown in Figure 4, for different levels of agility (dividing the project into two, five and nine parts, respectively). Note also that due to lower initial productivity, it is difficult for the team to finish the earlier iterations on time (assuming iterations have the same length and the same workload (Schatz and Abdelshafi 2005; Larman 2006; Lindstrom and Jeffries 2004). As a result, schedule pressure is higher during the earlier iterations than during the latter, leading to more overtime in the beginning of the project than in the end.

2.3.2 Exhaustion and turnover

When schedule pressure is persistently high *and* the period of overtime is long, negative effects can occur, as shown in Figure 5, exhaustion & turnover loop. Moore (2000)

found that technology professionals experiencing higher levels of exhaustion reported higher intentions to leave the job (Abdel-Hamid and Madnick 1991; Moore 2000; Oliva and Sterman 2001; Schatz and Abdelshafi 2005). Work overload contributed most to the level of exhaustion (Abdel-Hamid and Madnick 1991; Moore 2000; Oliva and Sterman 2001; Schatz and Abdelshafi 2005), compared to other possible factors such as role ambiguity, conflict, lack of autonomy or rewards. Note that turnover does not imply that a team member quits the organization. S/he may simply choose to be moved to another project.

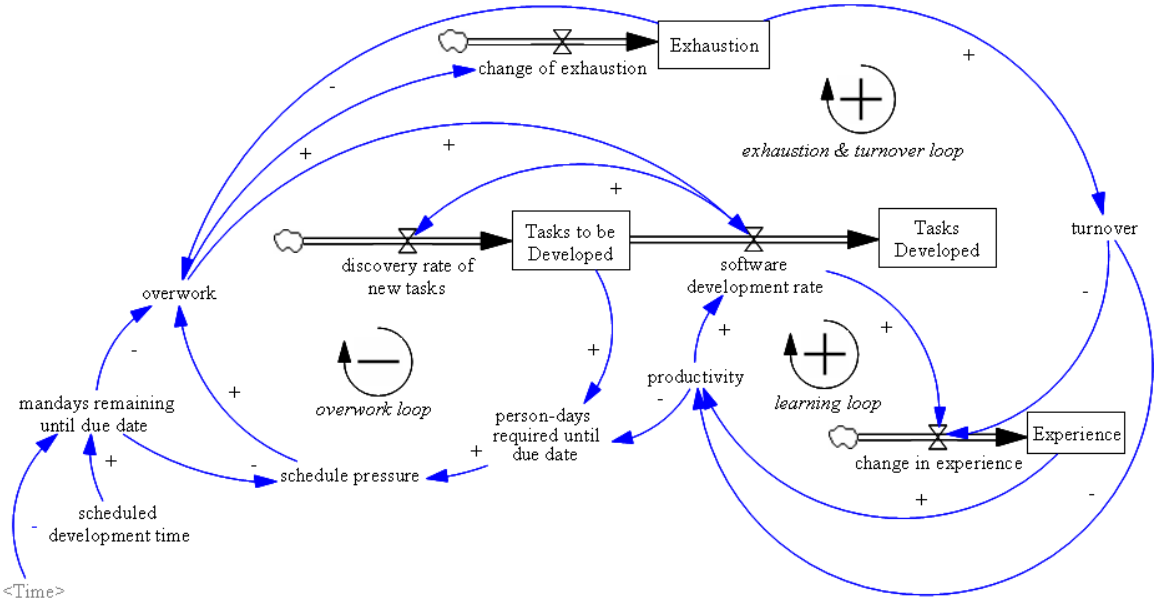

Figure 5: Negative effects of overwork

Exhaustion is modeled as a stock that increases each time the team works overtime (Figure 5). Exhaustion slowly builds up when the team works overtime to finish an iteration. As long as the team does not feel pressure in the next iteration, exhaustion gradually decreases (Abdel-Hamid and Madnick 1991). The level of exhaustion starts increasing when the team resumes working overtime. Beyond a point, working overtime may lead to exhaustion breakdown (burn-out). A breakdown occurs when the exhaustion level reaches a certain threshold. During the breakdown, the team is not willing (and not able) to work overtime anymore (Abdel-Hamid and Madnick 1991). Progress stagnates immediately, while schedule pressure keeps increasing. The shorter the time to decompress, the more vulnerable

the team is to an exhaustion breakdown (burn-out).

Attrition from exhaustion is modeled as increased turnover rate. Quits reduce team size immediately, thus leading to a decrease in team productivity (i.e., fewer tasks developed per day by the team). New employees hired as replacements usually have less experience than the experienced employees (Abdel-Hamid and Madnick 1991). Also, because experienced team members are the ones who train the newcomers (Brooks 1978; Abdel-Hamid and Madnick 1991), their productivity also suffers along with the new hires. Thus, hiring reduces the average experience level and productivity of the team.

Therefore, in addition to the two positive effects mentioned earlier, overtime can also causes two negative effects: exhaustion breakdown and/or increased turnover. These negative effects can form a vicious cycle that further increases schedule pressure. High pressure results in overtime, which causes increased exhaustion. Exhaustion may lead to a breakdown or increased turnover, further reducing productivity and the development rate. This can create a vicious cycle because an increase of schedule pressure causes team behavior that increases schedule pressure even more.

This is likely to be the situation for very low and very high levels of agility. A sequential project (Agile-1) has only one due date. The team works overtime continuously for a long time toward the end of the project, without time to decompress. At high agility levels, due dates follow each other so rapidly that the time available to de-exhaust is not enough to empty the exhaustion stock. Here, exhaustion builds up from one iteration to the next, causing a breakdown. In the worst case scenario, exhaustion leads to turnover, thereby reducing team size. Because of hiring and training delays (Brooks 1978; Sengupta and Abdel-Hamid 1996), team productivity is lower in iterations that immediately follow any turnover.

2.2.3 Errors and rework

Schedule pressure leads to an increased rate of errors (Abdel-Hamid and Madnick

1991; Zellmer-Bruhn 2003; Mass and Berkson 1995; Austin 2001; Oliva and Sterman 2001; Wickens 1984; Lyneis and Ford 2007; Austin 2001). Developers under pressure to meet a due date may take shortcuts in dealing with complications. Shortcuts are not in the best interests of the project, and usually have adverse effects that lead to extra (re)work later (Austin 2001; Oliva and Sterman 2001). Once detected, errors need to be fixed, thus increasing workload further. This creates a vicious cycle wherein high schedule pressure leads to higher error rates (Figure 6: errors & rework loop). The errors that are *detected* lead to more work, increasing schedule pressure even more. The errors that remain *undetected* result in a software product with lower quality (see also Austin, 2001).

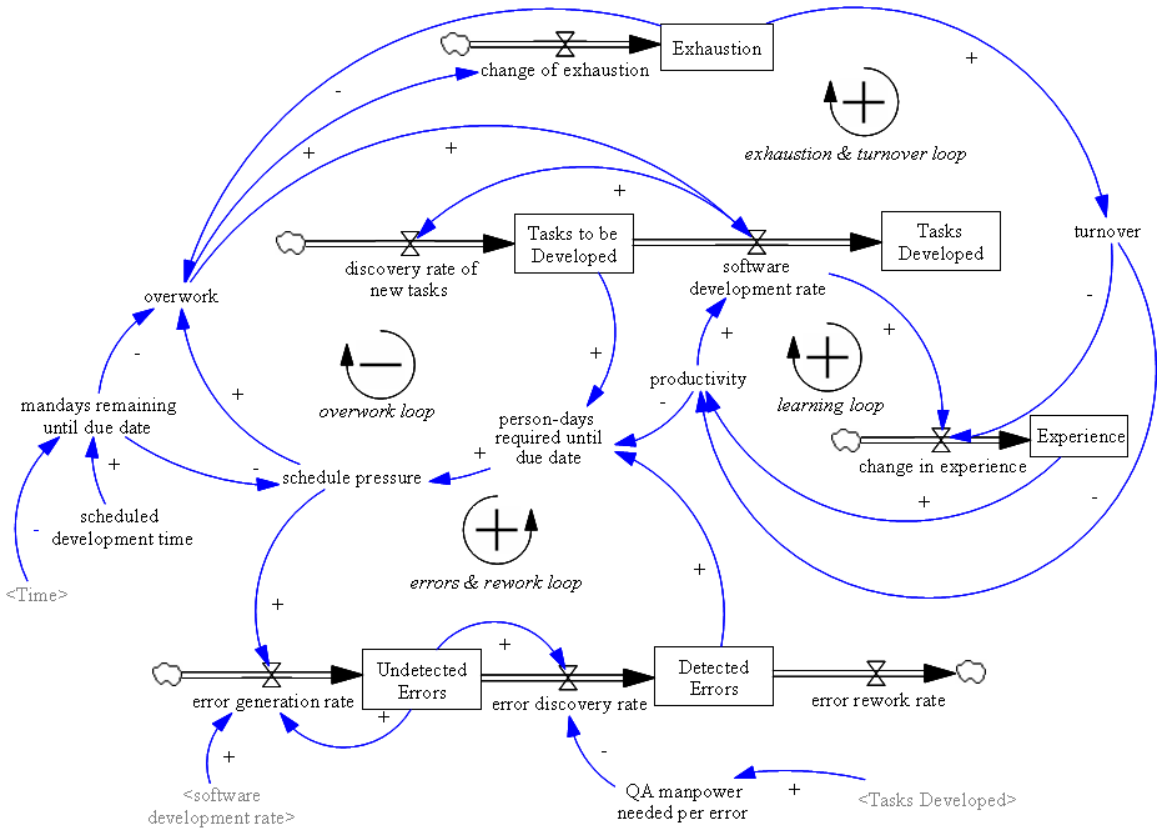

Figure 6: Negative effects of errors and rework

In sequential projects, more errors are generated toward the end of the project because of how pressure increases. In higher-agility levels, error generation is distributed throughout the project (i.e., during all pressure peaks, not just the last one). Rework on errors is therefore

also more balanced. However, here as well, rework increases workload and schedule pressure.

2.3 Summary

The model in Figure 6 shows that schedule pressure influences project outcomes in terms of quality (undetected errors), time and costs (person-days). Project outcomes may suffer either from a team being too inactive, e.g., in sequential or low levels of agility, but also from a team being over-active over too long, a situation likely to occur in high levels of agility. In both cases, the impacts on project outcomes are likely to be adverse. Thus, we hypothesize that *moderate levels of agility are likely to result in the best project outcomes.*

3 Method

We test this hypothesis through simulation/experimentation. Davis et al. (2007) argue that simulation should be the preferred approach for theory development when the research question involves a fundamental tension or trade-off among competing factors, such as the ones described in our hypothesis. The use of simulations also helps overcome the practical problems entailed in using other approaches for examining our research question. For example, it would be infeasible to ask software development teams to develop the same software product multiple times using different iteration lengths (or for that matter even ask different teams to develop the same product using different iteration lengths).

The simulations reported here use a system dynamics model of software development. System dynamics is especially suitable for situations involving multiple and interacting processes, time delays, and other nonlinear effects such as feedback loops and thresholds (Davis, et al., 2007). This modeling approach has been used extensively in similar research (e.g., Oliva and Sterman (2001); Repenning et al. (2001); Repenning and Sterman (2002); Lyneis and Ford (2007).

3.1 Simulation Model and Software Project

The model we use is an extensively validated model of software development (Abdel-Hamid and Madnick, 1991), and based on the figures shown in Section 2. The model has also been used as a research tool in several studies (e.g., Sengupta and Abdel-Hamid 1993; Sengupta et al. 1999; Abdel-Hamid et al. 1999), as well as in actual project settings in organizations (Sengupta et al. 2008). The model is a rich and widely accepted representation of software project dynamics, and is well-suited for testing our hypothesis. A brief description of the model is provided in the appendix.

The project used for the experiments is a simulation of a real-life organic project developed in-house for satellite telemetry software. The scheduled duration of the project is 260 working days when executed in a sequential mode (Agile 1). At the other end, the highest level of agility we examine consists of monthly iterative cycles (Agile 13 - 20 working days).

The project consists of 24,400 Delivered Source Instructions (DSI)[1]. However, because the full size of the project is not clear to the team in the beginning of the project, the perceived project size is smaller than the actual project size. In our simulation model we assume that 35% of all tasks that need to be done are unknown at the start of the project. This is similar to the level of uncertainty that was used in the model of Abdel-Hamid and Madnick (1991). The level conforms to Boehm's estimates as well.

We use the COCOMO equations to estimate the perceived project size in person-days from the number of DSI that the project should deliver [2] (Boehm 1981; Abdel-Hamid and

[1] We also conducted experiments with a smaller project (12200 DSI). The results are substantially the same as those reported here.

[2] *Planned person-days for project = 2.73*19*(perceived project size in DSI/1000)$^{1.05}$*
*Perceived project size = actual project size in DSI * uncertainty fraction*
The perceived project size = 24400 * 0.65 = 15860 DSI, consequently, the planned person-days for the project = 2.73*19*(15.860)$^{1.05}$ = 945 person-days. The project should be finished in 260 days (comparable to the target development time of the software development project that was simulated by Abdel-Hamid and Madnick, 1991). The workforce is set to 4.9 persons. With this initial workforce the team perceives it has sufficient time for the project (945/4.9 < 260). However, when all tasks are known, this workforce turns out to be too small. For a project of 24400 DSI at least 1485 person-days are required. With a team of 4.9 persons this leads to a development time of 303 days. So, with a team of 4.9, schedule pressure increases and overwork is required.

Madnick 1991). The minimal required size of the project team can be determined from the perceived project size (in person-days) and the scheduled development time. Thus, when perceived project size is 1000 person-days and the scheduled development time 200 days, a team of 5 employees is required.

The model of Abdel-Hamid and Madnick (1991) is based on a sequential software development model. Therefore, we had to adjust the model for our research purposes. Our operationalization of agile development follows Boehm and Turner's (2005) recommendation for partitioning a project into multiple parts of similar iteration lengths (Agile-n) that are executed sequentially. When all parts are developed, the complete project is also finished. The stocks and flows of the agility subsystem are shown in Figure 7.

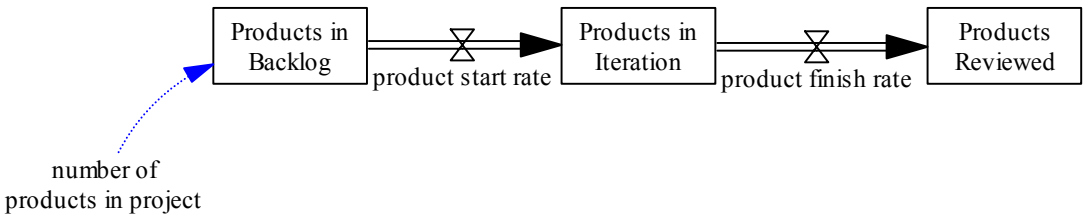

Figure 7: Stocks and flows of the agile subsystem

The level of agility determines how many subsequent products or parts will be developed. So, when the level of agility is 5, the project is divided into 5 parts or iterations and 5 intermediate products will be developed. All 5 products are assumed to be of equal size that is given by the total project size divided by 5. The initial values of the stocks and flows and the equations that are used in the agile subsystem are given in Table 1. At $t = 1$, the model starts simulating the development of the first part or product. As soon as the team is finished with the first product, the product finish rate is 1, and the team begins developing the next product. In summary, each time the stock 'Products in Iteration' is 1, the simulation model will run again, until all agile project parts are finished (Products Reviewed = number of products in project).

Table 1: Equations for the agile subsystem

Nr	Equation
Eq 1	Products in Backlog = number of products in project - 1
Eq 2	Initial values: - Product start rate = 0 - Products in Iteration = 1 - Product finish rate = 0 - Products Reviewed = 0
Eq 3	Real product size in DSI = real project size in DSI / number of products in project (24400/5=4880 DSI)
Eq 4	Perceived product size in DSI = real product size in DSI * (1 – uncertainty%)
Eq 5	Planned person-days of product = planned person-days of project/ number of products in project
Eq 6	Planned development time of product = planned development time of project / number of products in project
Eq 7	Product finish rate = IF THEN ELSE (actual product% complete=1,Products in Iteration,0)
Eq 8	Actual product% complete = Tasks Finished / real product size in tasks
Eq 9	Product start rate = IF THEN ELSE (Products in Iteration=0,MIN(1,Products in Backlog),0)

3.2 Scenarios

We implemented two scenarios which differed with respect to whether customers choose to change requirements between iterations. This often occurs during reviews between iterations. After reviewing the functionality developed during an iteration, a customer may decide to make requests for modifications during the next iteration (e.g., a change in specifications for the look and feel of a screen). The requests for modification require changes made in what was previously developed.

We operationalize this aspect of agile projects, at two levels. The *No Change* level assumes that customers do not change their requirements; and that a part developed during an iteration does not require any extra effort/rework of the parts already developed. (Note: there can still be undiscovered tasks that come to attention when executing known tasks).

In the level incorporating *Requirements' Changes*, the development of each new part requires additional effort in modifying existing parts. The number of extra tasks thus created is assumed to be a function of the number of iterations: ½ * $n(n-1)$, where n is the number of iterations (Brooks 1979). Thus, when a project has 4 iterations, 6 extra tasks need to be done by the team because customers have changed requirements.

3.3 Dependent Variables

Our primary outcome variable is the *number of undetected errors* remaining, because of how the level of agility affects the peaks of schedule pressure, and therefore quality. A software product with many undetected errors (bugs) in the code is assumed to be of inferior quality than a product with few undetected errors. Additionally, the errors that are detected require effort to fix, and thus also impact two other outcome measures widely used in projects: *development time* (number of working days needed to complete the project) and the *number of person-days expended* (including any overtime expended).

S*chedule pressure* is our key process variable. Schedule pressure is analyzed in two different ways. A*verage schedule pressure* for a period is calculated as the sum of daily schedule pressure during the period divided by the number of days in that period. Second, the *behavior over time of daily schedule pressure* is analyzed during the project (to analyze the pattern of oscillation).

4 Results

4.1 Project outcomes

Table 2 shows the project outcomes in terms of quality, effort and time for the two scenarios described above: *No Change*, i.e., assuming that customers do not change their requirements between iterations, and *Requirements Change*, assuming that they do. The differences between levels of agility with respect to actual development *time* are relatively small (column 5 & 8). This is because adherence to schedule is the first goal that the simulation model tries to accomplish. Therefore, scenarios differ more with respect to *quality* (undetected errors, column 3 & 6) and *cost* (person-days expended, column 4 & 7).

In both scenarios, performance improves with the level of agility. The numbers of undetected errors decrease, as do effort and time. However, this occurs only up to Agile-4 to Agile-6, and reverses at higher levels. In other words, there appears to be a U-shaped trend in

performance. In the *requirements change* scenario, the reversal in performance is even more severe for higher levels of agility. To understand the reasons underlying the U-shaped trend in performance, we examine how iteration lengths drive the core underlying mechanism schedule pressure, and in turn, outcomes.

Table 2: Project outcomes

Agility	Iteration (days)	No change			Requirements change		
		Undetected errors	Effort (person days)	Time (days)	Undetected errors	Effort (person days)	Time (days)
Agile 1	260	239	1361	280	239	1361	280
Agile 2	130	198	1291	262	201	1292	262
Agile 3	87	179	1286	261	189	1287	261
Agile 4	65	164	1285	261	183	1293	262
Agile 5	52	160	1291	262	191	1294	262
Agile 6	43	164	1292	262	206	1296	262
Agile 7	37	174	1293	262	274	1307	265
Agile 9	29	189	1311	264	401	1349	285
Agile 11	24	272	1306	263	595	1325	268
Agile 13	20	416	1302	264	839	1390	284

4.2 Schedule pressure and progress

Figure 8 shows the average schedule pressure during the project for each level of agility in the two scenarios. Figure 9 shows the behavior of schedule pressure over time for different levels of agility for the No Change scenario. Some agility levels are not reported in the figures to facilitate visual exposition.

The average schedule pressure across various levels of agility (Figure 8) shows a U-shaped distribution similar to the outcomes. Figure 9 indicates that for each agility level, schedule pressure has a low initial value at the start of an iteration, and peaks at the end. However, the heights and durations of the peaks differ both within and across different agility levels. In Agile-1, it takes 137 days (more than 50% of the scheduled development time) before the team realizes that the time remaining is less than the time required. Schedule pressure never drops after that day, and accumulates rapidly. In these first 137 days, the team

has underestimated total development effort and has underutilized time. By the time the team realizes that it needs to work much harder, it is too late. The result is low schedule pressure for much of the project, which leads to working "undertime": inefficiently or slowly, followed by a steep rise. Note that the rise is even steeper after day 211. This is the time the team has an exhaustion breakdown after working overtime for too long. As of this day the team only works normal hours, with the expected negative effect on schedule pressure.

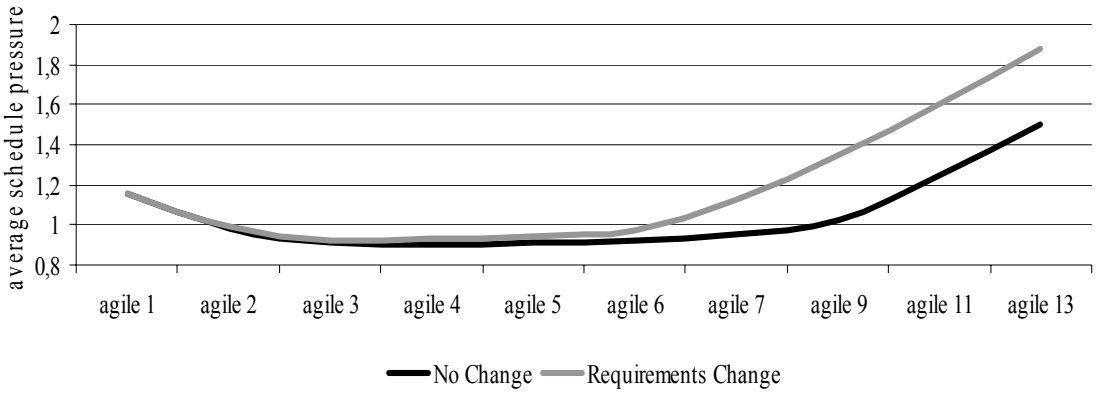

Figure 8: Average Schedule Pressure

Now consider what happens when the due date for the first deliverable arrives earlier than in Agile-1, e.g., Agile-6[3]. In order to complete the iteration on time, the team needs to make progress from the beginning. Thus, schedule pressure starts earlier: in Agile-6: it takes the team only 33 days to notice that the time required to finish the iteration is smaller than the time remaining (Figure 9). However, as Figure 8 shows, the average schedule pressure is *lower* in Agile-6 (for both situations) than in Agile-1.

[3] Agile-6 is chosen here as an example because this level of agility lies between the two extremes: Agile-1 and Agile-13, and furthermore, Agile-6 has much better project outcomes than Agile-1 and Agile-13 (Table 2).

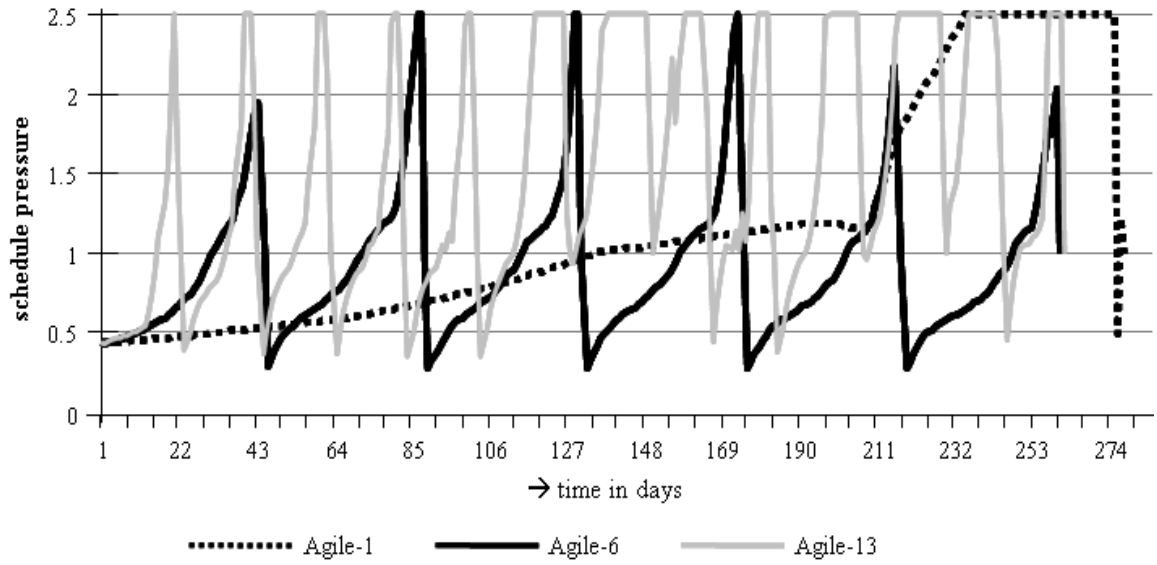

Figure 9: Schedule pressure during the project in No Change Scenario

Thus, up to Agile-6, the effect of increasing agility on schedule pressure is two-fold: pressure is perceived earlier, but has less of a cumulative impact, because on average schedule pressure is lower. The result is the downward path of average schedule pressure for Agile-1 to Agile-6 shown in Figure 8. Therefore, with higher agility less time is wasted at the start, more work gets done early and less work needs to be done at the end (also shown by the lower schedule pressure peaks at the end of the Agile-6 project in Figure 9).

This activates a virtuous cycle of learning and thus, productivity. When agility increases from Agile-1 to Agile-6, we see an increase in schedule pressure, which triggers productivity (via the *overwork & learning loop*). This increase in productivity is shown in Figure 10: to develop for example, the first 100 tasks, in the Agile-6

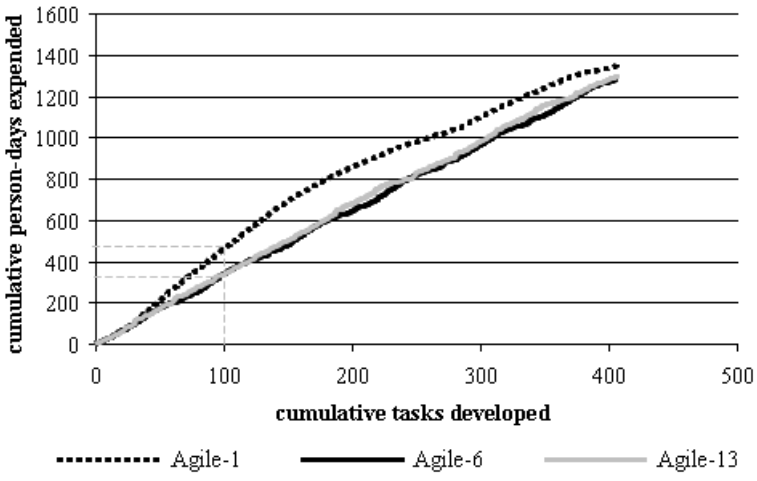

Figure 10: Productivity in the No Change situation

scenario 340 person-days are required, as opposed to 464 person-days in the Agile-1 scenario.

This further helps the team manage workload in the later parts of the project. The team is already productive, and good progress ensures that there is less continuing schedule pressure and therefore the need for overwork. This is shown in the contrasting patterns in schedule pressure in the later stages between Agile-1 and Agile-6 in Figure 9. The final implication is that the Agile-6 team is less pressed for time than the Agile-1 team, and thus less prone to take shortcuts that stimulate the error-generation and rework cycle discussed in Figure 1. This is shown by the lower number of undetected errors for Agile-6 compared to Agile-1 (Table 2).

Now consider what happens with further increases in agility from Agile-6 to Agile-13. Higher levels of agility imply earlier due dates, and require even earlier progress. Moreover, progress is not linear: productivity is low at the beginning of the project and the learning curve takes time. Thus even though the amount of work required for the first deliverable is lower in higher levels of agility (7.7% for Agile-13 compared with 16.7% for Agile-6), the time available is also proportionately lower. Consequently, teams with higher levels of agility are forced to spend more effort to get the work done initially, and experience higher schedule pressure. Expectedly, with increase in agility, the schedule pressure starts earlier. For example, Figure 9 shows that the schedule pressure for the highest level of agility considered here (Agile-13) starts earlier than for any other level (on day 18 schedule pressure rises above 1 for the first time). However, in contrast to, e.g., Agile-6, the pressure also accumulates to increasingly higher levels. The combined effect is the upward drift of average schedule pressure from Agile-6 to Agile-13 (Figure 8). The drift is even more aggressive in the situation where Requirements Change.

In the Requirements Change situation the *average schedule pressure* for lower levels

of agility (Agile-1 to Agile-4) are almost similar to those of the No Change situation (Figure 8). However, looking at the results on quality in Table 2 we see that as of Agile-3 the number of undetected errors is much higher than in the No Change situation. High levels of agility result in much more errors due to the changing requirements and the accompanying extra workload. Since there is hardly any time for the extra work packages that need to be done in this scenario, schedule pressure is increased even more than in the No Change situation, leading to more errors. In Figure 11 the behavior over time of three variables is shown in both the No Change and Requirement Change situation for both Agile-6 and Agile-13.

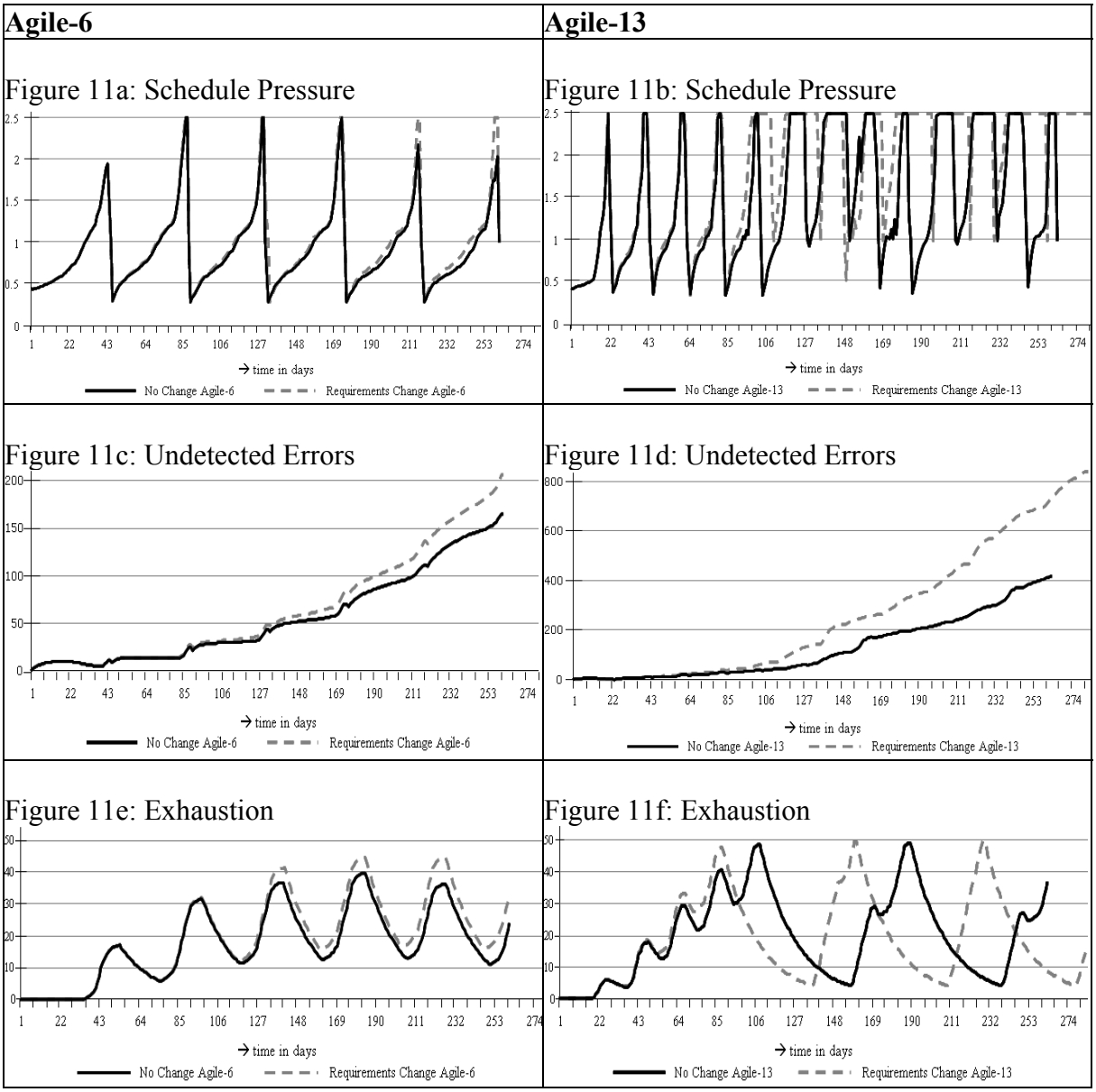

Figure 11: Schedule pressure, undetected errors and exhaustion for Agile-6 & -13

In the Requirements Change situation (compared to No Change) schedule pressure increases slightly in the Agile-6 scenario, but severely in Agile-13 (Figure 11a & b). Because of the increased pressure the team is under, more errors are made, and less errors are detected, leading to a huge increase in the level of undetected errors (Figure 11c & d, note that the scale in (Figure 11d is different from that in Figure 11c). Increased schedule pressure leads to working overtime which leads to an increase of the exhaustion of the team. The exhaustion in the Agile-6 scenarios remains under the exhaustion breakdown threshold. This means that at the end of an iteration, the team has sufficient time to recover from the pressure and to decompress before the pressure of the next iteration hits them (Figure 11e). In the Agile-13 scenarios this is not the case. Here, the team suffers from exhaustion breakdowns several times during the project (2 times in the No Change situation and even 3 times in the Requirement Change situation). After an exhaustion breakdown, the team is not willing to work overtime until the team is de-exhausted. This takes on average 40 working days. During this time no extra progress is made, keeping schedule pressure high (with its negative effects on errors). So, the graphs in Figure 11 and the results in Table 2 show that when requirements can change, the performance differences between mid- and high levels of agility become much more profound, in favor of the mid-levels.

Thus, beyond a point (Agile-4 to Agile-6 in our simulation), having increasingly earlier deadlines for the first few deliverables (as is implied by higher levels of agility) becomes relatively inefficient because it takes time for the learning effect to occur (and because of the extra workload that occurs in the Requirements Change situation). For example, consider how many person-days are required to finish the first 100 tasks in the Agile-6 and the Agile-13 scenario (Figure 10). As was described earlier, in the Agile-6 scenario 340 person-days are required, as opposed to 342 person-days in the Agile-13 scenario.

The first iteration in the Agile-13 project is finished 1 day too late. Because iteration lengths are equal, the lateness of the first iteration leads to a reduction of the available time for the second iteration. Thus, at the start of the second iteration, the team faces even less time available. Due to the lateness of the first iteration, the second iteration is almost as difficult for the team to finish on time, even when learning and productivity have increased. Because of the need to catch-up on the lateness of the first iteration(s), teams in highly agile projects still experience high schedule pressure peaks at the end of the project (see Figure 9). Thus, higher levels of agility are likely to have delays in subsequent iterations, resulting in late projects.

Summarizing, the relatively poor performance of the higher agility scenarios is caused primarily by *inefficiency due to over-activity*. The early due dates are simply planned too early. Problems from the first iteration cascade to the next iteration. In the Agile-6 project, a future iteration is easier to finish than the previous one. In the Agile-13 project, a future iteration is almost as difficult to finish as the previous one. The team is constantly forced to make more progress than they can handle (with the given productivity), leading to an overly active team that needs to work overtime very often to make up for the low productivity and the lateness of the first iteration(s).

The effect of over-activity is aggravated for the Requirements Change situation. Here, apart from the tasks remaining from the previous iteration, the team also needs to do some unexpected extra tasks that were discovered during the feedback session with the customer. As a result schedule pressure increases even more (see Figure 8), and project outcomes suffer (Table 2).

5 Discussion and Conclusion

Many managers and organizations have increasingly favored iterative approaches such as agile development to improve the effectiveness and efficiency of projects. While agile approaches have many flavors, practitioners generally anchor their projects on a 20 working-day monthly iterative cycle of development and customer feedback. However, there is no sufficient evidence of the effectiveness of this 20 working-day cycle. This is the essence of our research question.

Project outcomes under different levels of agility are influenced by schedule pressure. Long iterations lengths (e.g., sequential development) create little schedule pressure initially, leading to inefficiency caused by inactivity. Slicing the project into smaller parts (i.e. smaller iteration lengths) brings the next delivery date progressively closer, thus providing a sense of urgency and creating greater schedule pressure. Higher schedule pressure leads to more work being accomplished, in turn enhancing the experience cycle, thereby improving productivity.

However, it takes time for the experience cycle (and therefore higher productivity) to kick in. Thus, beyond a point, compressing iteration lengths simply adds to schedule pressure without commensurate increases in experience (and productivity). Progress is forced to occur too quickly while productivity is still low, leading to overwork (over-activity). The team gets further behind in deliverables; and exhaustion from high levels of schedule pressure reduces productivity. Attrition causes schedule pressure to increase further, leading to less attention to error detection and correction, higher costs, and more development time. The pattern is accentuated when customers are able to change requirements between iterations.

Our simulations show that moderate levels of agility have the best performance (Agile-4 - Agile-6). The best options for projects are to work in slices of 43-65 working days, i.e., around two to three calendar months. Shorter iterations – e.g. the 20 working days-iterations that are often advocated by practitioners of agility (e.g., the Scrum process - Schatz

and Abdelshafi, 2005; or Larman (2006) recommendation of 5-30 working days) – appear to be less beneficial to project outcomes.

Thus, there is a boundary beyond which agility may not be useful. Shortening the length of the iteration any further is unlikely to improve performance, and may well decrease it. Furthermore, the iteration lengths of the best levels of agility are much longer than those prescribed in the literature and widely adopted by practitioners.

In our study (as in agile development), the customer is able to specify requirements for the next phase regardless of the length of the iteration. This means that long iteration lengths are only possible when the customer is able to determine what deliverables should be developed. In practice, there should be some alignment between the interval at which new requirements/change requests can reasonably be expected to occur, and the interval of reviews. In environments where specifications are highly volatile, requirements can only be specified for short intervals, e.g., one month. In these cases, iteration lengths of 45-54 days may not be appropriate. Such projects need great flexibility, and the project manager may instead be better off with monthly intervals, even if the efficiencies are lower. However, that does not imply that *all* development situations need monthly cycles. Shorter cycles can arguably foster a lack of discipline in requirements generation. Some requirements and modifications are likely to be specified simply because of the flexibility available from short cycles. In such cases, high agility can potentially destroy value. Evidence shows that organizations with high levels of development capability (CMM level 5) exercise process maturity by finding ways to *limit* the number of modifications that customers make (Agarwal and Chari 2007), typically by working with customers to help them narrow the pool of potential requirements/changes. Our finding can be thought of as a temporal version of this argument: moderation in agility can improve project performance by limiting the number of

modifications (and any cascading effect), while at the same time providing flexibility when needed.

References

Abdel-Hamid, Tarek K., and Stuart E. Madnick. 1991. *Software Project Dynamics – an integrated approach.* Prentice Hall, Englewood Cliffs, New Jersey.

Abdel-Hamid, Tarek K., Kishore Sengupta, and Clint Swett. 1999. The Impact of Goals on Software Project Management: An Experimental Investigation. *MIS Quarterly.* 23:4:531-555.

Agrawal, Manish, and Kaushal Chari. 2007. Software Effort, Quality, and Cycle Time: A Study of CMM Level 5 Projects. *IEEE Transactions on Software Engineering*. 33:3:145-156.

Austin, Robert D. 2001. The Effects of Time Pressure on Quality in Software Development: An Agency Model. *Information Systems Research*. 12:2:195-207.

Berger, Hilary. 2007. Agile development in a bureaucratic arena - A case study experience. *International Journal of Information Management.* 27:386-396.

Buehler, Roger, Dale Griffin, and Michael Ross. 1994. Exploring the "planning fallacy": why people underestimate their task completion times. *Journal of Personality and Social Psychology*. 67: 366-381.

Boehm, Barry W. 1981. *Software Engineering Economics*. Englewood Cliffs, New Jersey: Prentice-Hall, Inc.

Boehm, Barry, and Richard Turner. 2005. Management Challenges to Implementing Agile Processes in Traditional Development Organizations. *IEEE Software*. 22:5:30-39

Boh, Wai Fong, Sandra A. Slaughter, and J. Alberto Espinosa. 2007. Learning from

Experience in Software Development: A Multilevel Analysis. *Management Science*. 53:8:1315-1331.

Brooks Jr., Fred P. 1979. *The Mythical Man-Month, Essays on Software Engineering*, University of North Carolina, Chapel Hill, Addison-Wesley Publishing Company.

Cao Lan, and Balasubramaniam Ramesh. 2008. Agile Requirements Egnineering Practices: An Empirical Study. *IEEE Software*. 60-67.

Davis, Jason P., Kathleen M. Eisenhardt, and Christopher B. Bingman. 2007. Developing Theory through Simulation Methods. *Academy of Management Review*. 32:2:480-499.

Dawson, Ray J., and Christian W. Dawson. 1998. Practical proposals for managing uncertainty and risk in project planning. *International Journal of Project Management*. 16:299-310.

Eisenhardt, Kathleen M., and Behnam N. Tabrizi. 1995. Accelerating adaptive processes: product innovation in the global computer industry. *Administrative Science Quarterly*. 40:84-110.

Ford, David N., and John D. Sterman. 1998. Dynamic modeling of product development processes. *System Dynamics Review*. 14:1:31-68.

Gevers, Josette M.P., Wendelien van Eerde, and Christel G. Rutte. 2001. Time Pressure, Potency, and Progress in Project Groups. *European Journal of Work and Organizational Psychology*. 10:2:205-221.

Goldratt, Eliyahu M. 1997. *Critical Chain*. The North River Press, Great Barrington, MA.

Iansiti, Marco, and Alan MacCormack. 1997. Developing Products on Internet Time. *Harvard Business Review*. 108-117.

Karau, Steven J., and Janice R. Kelly. 1992. The effects of time scarcity and time abundance on group performance quality and interaction process. *Journal of Experimental Social*

Psychology. 28:542-571.

Kessler, Eric H., Paul E. Bierly. 2002. Is faster really better? An empirical test of the implications of innovation speed. *IEEE Transactions on Engineering Management*. 49:1:2-12.

Larman, Craig. 2006. *Agile and Iterative Development, a manager's guide*. Agile Software Development Series. Edited by Alistair Cockburn and Jim Highsmith. Addison-Wesley Publishing Company.

Latham, Gary P., and Gerard H. Seijts. 1999. The effects of proximal and distal goals on perforamnce on a moderately complex task. *Journal of Organizational Behavior*. 20:421-429.

Lindstrom, Lowell, and Ron Jeffries. 2004. Extreme Programming and Agile Software Development Methodologies. *Information Systems Management*. 21:3:41-60.

Lyneis, James M, and David N. Ford. 2007. System Dynamics Applied to Project Management: a survey, assessment, and directions for future research. *System Dynamics Review*. 23:157-189.

Mass, Nathaniel J., and Brad Berkson. 1995. Going Slow to Go Fast. *The McKinsey Quarterly*. 4:19-29.

McDermott, Christopher M. 1999. Managing radical product development in large manufacturing firms: a longitudinal study. *Journal of Operations Management*. 17:631-644.

Meso, Peter, and Radhika Jain. 2006. Agile Software Development: Adaptive Systems Principles and Best Practices. *Information Systems Management*. 23:3:19-39.

Molokken-Østvold, Kjetil, and Magne Jørgensen. 2005. A Comparison of Software Project Overruns-Flexible versus Sequential Development Models. *IEEE Transactions on*

Software Engineering. 31:9:754-766.

Moore, Jo E. 2000. One road to turnover: an examination of work exhaustion in technology professionals. *MIS Quarterly*. 24:1:141-168.

Oliva, Rogelio, and John D. Sterman. 2001. Cutting Corners and Working Overtime: Quality Erosion in the Service Industry. *Management Science*. 47:7:894-914.

Oorschot, Kim E. van, Will J.M. Bertrand, and Christel G. Rutte. 2005. Field studies into the dynamics of product development tasks. *International Journal of Operations and Production Management*. 25:8:720-739.

Perlow, Leslie A. 1999. The Time Famine: Toward a Sociology of Work Time. *Administrative Science Quarterly*. 44:57-81.

Repenning, Nelson P., Paulo Goncalves, and Black, L.J. 2001. Systems Dynamics - Past the Tipping Point: - The Persistence of Firefighting in Product Development. California Management Review. 43:44-63.

Repenning, Nelson P., and John D. Sterman. 2002. Capability Traps and Self-Confirming Attribution Errors in the Dynamics of Process Improvements. *Administrative Science Quarterly*. 47:265-295.

Schatz, Bob, and Ibrahim Abdelshafi. 2005. Primavera Gets Agile: A Successful Transition to Agile Development. *IEEE Software*. 22:3:36-42.

Seers, Anson, and Steve Woodruff. 1997. Temporal pacing in task forces: group development or deadline pressure. *Journal of Management*. 23:169-187.

Sengupta, Kishore, and Tarek K. Abdel-Hamid. 1993. Alternative conceptions of Feedback in Dynamic Decision Environments: An Experimental Investigation. *Management Science*. 39:411-428.

Sengupta, Kishore, and Tarek K. Abdel-Hamid. 1996. The Impact of Unreliable Information

on the Management of Software Projects: A Dynamic Decision Perspective. *IEEE Transactions on Systems, Man, and Cybernetics*. 26:177-189.

Sengupta, Kishore, Tarek K. Abdel-Hamid, and Michael Bosley. 1999. Coping with Staffing Delays in Software Project Management: An Experimental Investigation. *IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans*. 29:1:77-91.

Sengupta, Kishore, Tarek K. Abdel-Hamid, and Luk N. Van Wassenhove. 2008. The Experience Trap. *Harvard Business Review*. 86:2:94-102.

Seshadri, Sridhar, and Zur Shapira. 2000. Managerial Allocation of Time and Effort: The Effects of Interruptions. *Management Science*. 47:5:647-662.

Sterman, John D. 2000. *Business Dynamics - Systems Thinking and Modeling for a Complex World*. Irwin McGraw-Hill, Boston.

Tatikonda, Mohan V., and Stephen R. Rosenthal. 2000. Successful execution of product development projects: balancing firmness and flexibility in the innovation process. *Journal of Operations Management*. 18:401-425.

Thaler, Richard H. 1999. Mental Accounting Matters. *Journal of Behavioral Decision Making*. 12:183-206.

Wickens, Christopher D. "Processing resources in attention." In *Varieties of Attention*, edited by R. Parasuraman and D.R. Davies. 63-101. Orlando Florida: Academic Press Inc., 1984.

Wiersma, Eelke. 2007. Conditions that Shape the Learning Curve: Factors that Increase the Ability and Opportunity to Learn. *Management Science*. 53: 12:1903-1915.

Zellmer-Bruhn, Mary E. 2003. Interruptive Events and Team Knowledge Acquisition. *Management Science*. 49:4:514-528.